Diverse Generation from a Single Video Made Possible (Supplementary Material)

This is a part of the full supplementary HTML which contains all videos.

1 Implementation Details

All RGB values in the videos are scaled to $[-1, 1]$. All runs were conducted on Quadro RTX 8000 GPU.

1.1 Creating the Spatio-Temporal Pyramid

Given a downscaling factor r (where $r = (r_H, r_W, r_T)$) and a minimal size m_S, m_T, we keep downscaling the input video in all dimensions until it “hits” the minimal size in the spatial or temporal dimensions. Assume we hit the minimal spatial dimension first, we keep on downscaling the temporal dimension until reaching its minimal size, while keeping the spatial dimensions fixed on its minimal size (and the opposite goes if we first hit the temporal dimension, keeping on downscaling the spatial dimensions while keeping the temporal fixed). The minimal size of the spatial dimensions, m_S, is the minimum between both height and width (namely, no spatial dimension will be smaller than m_S).

We use cubic downscaling interpolation for both temporal and sptial dimensions. We tried to use nearest interpolation on the temporal dimension, because it might make more sense sometimes, but found that in most applications it performed the same or worse.

1.2 Diverse Generation Technical Details

The input to the coarsest scale is $(z_N + x_N) \sim \mathcal{N}(x_N, \sigma\mathbb{I})$ where σ is a hyperparameter, and we use $\sigma \in [2, 5]$. a good "thumb rule" for σ is the mean distance between each patch to its nearest neighbour (other than itself). Downscaling factor is 0.82 for the spatial dimensions (height and width) and 0.87 for the temporal dimension. The minimal size of the pyramid is set to 3 frames with minimal spatial dimension of 15 pixels. We use patch size $(3 \times 7 \times 7)$, where 3 is in the temporal dimension. We use 5 EM-like iterations in each scale of the pyramid. When the number of voxels $(T \times H \times W)$ is larger than $3,000,000$ we change the number of EM-like iterations to 1, and the patch-size to $(3 \times 5 \times 5)$. This change reduce runtime without hurting the quality of the results.

1.3 Video Analogies Technical Details

For all examples we use patch size $(3 \times 5 \times 5)$ and $\alpha = 1$ (for completeness score). For all-pairs examples we use downscaling factor is 0.9 for all dimensions. The minimal size of the pyramid is set to 3 frames with minimal spatial dimension of 20 pixels. 1 EM-like iterations per scale. For sketch-to-video examples we use downscaling factor of 0.78 for all dimensions and minimal size is 5 frames and minimal spatial dimension of 35. 3 EM-like iterations per scale (and 1 for the last two scales, to save runtime). Runtime per result is about 1 minute.

1.4 PatchMatch Implementation Details

In all applications we use mean-square-error as the distance function between. In Fig. 1 we show another more detailed comparison between our PatchMatch implementation and the exhaustive nearest-neighbor search used by GPNN. Our implementation has time complexity of $O(n \times d)$ and $O(n)$ additional memory (where n is the video size and d is the patch size), compared to GPNN, with time complexity of $O(n^2 \times d)$ and memory footprint $O(n \times d)$. This is easily seen in the figure. We use the same propagation and random search steps as in the original PatchMatch paper [1], using the "jump flood" scheme [3]. In each PatchMatch iteration we look at 4 neighbors at distance *step* (with additional small noise for the exact position of the neighbor, and without) and a random search. However, we only use 15 PatchMatch iterations per VPNN usage, this is done by searching for $step = 8, 4, 1$ 5 times.

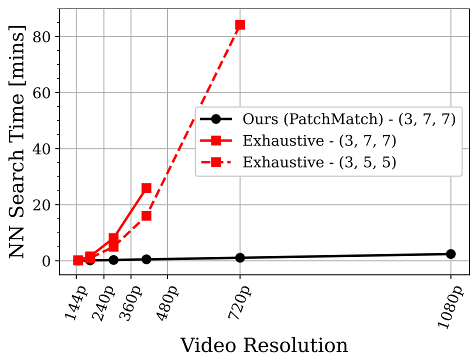


Fig. 1: **Nearest Neighbor Search Comparison** using PatchMatch (in our method) vs. exhaustive search (used by GPNN [2]). GPNN exceeds GPU memory at medium resolution (480p) with the original patch size $(3, 7, 7)$. The dashed line with smaller patch size $(3, 5, 5)$ is intended to show the quadratic trend with more data points.

2 Further Properties

2.1 Importance of patch-size and size of pyramid

Using larger patch-size is equivalent to adding levels to the pyramid (keeping the down-scale fixed), going to a smaller size in the coarsest level. Both ways cause patches to "capture" larger regions from the original input, but smaller patch-size is preferred due to computation time. The main hyperparameters are therefore the size of the coarsest level, and the std of the noise. These are more like "knobs" that affect the nature of the results.

2.2 Layout-Appearance Tradeoff in Video Analogies

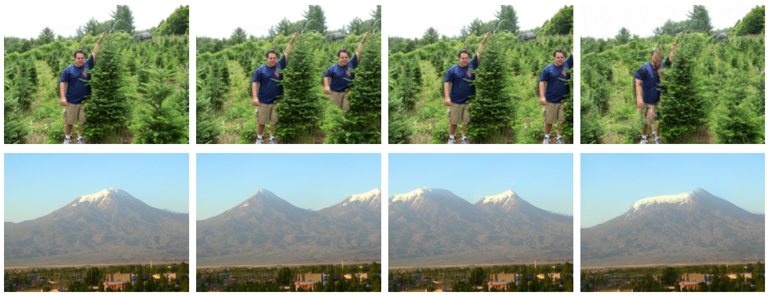
Since we are trying to create a new video whose spatio-temporal layout (modeled with the dynamic structure) taken from one video and its appearance from another, there's an inherent tradeoff of which of the two we want to be better preserved in the result. The dynamic structure is "enforced" by using the auxiliary channels in Q and K. Removing these channels would generate a video that is much more similar in its appearance to S but bears less resemblance to the spatio-temporal layout of C. We can control this tradeoff by setting an upper limit in the pyramid from which we stop using the auxiliary channels. In our results it was best to set the maximal scale at half the pyramid height.

3 Comparison details

Evaluation Set Details. For each input video in HP-VAE-GAN and SinGAN-GIF datasets we generated the same number of random sample as publicly available (10 generations for each video in HP-VAE-GAN dataset, and 6 generations for each video in SinGAN-GIF dataset), and compared their SVFID and diversity.

Quality Comparison to GPNN [2]. We provide further evidence that, other than the gain in speed (shown in Fig.5 in the paper), using PatchMatch [1] does not result in loss of quality. We compare to the same image dataset used by SinGAN [4] and GPNN [2] by generating 50 samples for each of the 50 inputs (using our approach with patch-size=1 in the temporal dimension). In this experiment, the only difference between us and GPNN [2] is the use of PatchMatch. As seen in the Table below, the quality of our results (in terms of SIFID [4]) is similar to GPNN, while keeping similar diversity. This can also be seen visually in the Figure below (all generated samples are in the supplementary files).

	SIFID $\downarrow$	Diversity	SIFID $\downarrow$	Diversity
SinGAN [4]	0.051	0.35	0.085	0.5
GPNN [2]	0.030	0.35	0.077	0.47
Ours	0.026	0.33	0.065	0.47



Video Diversity Index. The video adaptation of the *diversity* index (originally proposed for images by [4]) is: given an input video, the standard deviation of each video position (3D RGB element in the video, converted to grayscale) is calculated across all generated samples, and then averaged across all pixels. This is then divided by the standard deviation of the voxels in the input video.

Bibliography

[1] Connelly Barnes, Eli Shechtman, Adam Finkelstein, and Dan B Goldman. Patchmatch: A randomized correspondence algorithm for structural image editing. *ACM Trans. Graph.*, 28(3):24, 2009.

[2] Niv Granot, Ben Feinstein, Assaf Shocher, Shai Bagon, and Michal Irani. Drop the gan: In defense of patches nearest neighbors as single image generative models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13460–13469, 2022.

[3] Guodong Rong and Tiow-Seng Tan. Jump flooding in gpu with applications to voronoi diagram and distance transform. In *Proceedings of the 2006 symposium on Interactive 3D graphics and games*, pages 109–116, 2006.

[4] Tamar Rott Shaham, Tali Dekel, and Tomer Michaeli. Singan: Learning a generative model from a single natural image. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4570–4580, 2019.